Getting Software Engineering into Our Guts

Lawrence Bernstein and David Klappholz
Stevens Institute of Technology

Many if not most, computer science students are enamored of technology (state of the art), but averse to the discipline of software process (state of the practice). Staying up late hacking code and eating pizza is great fun for them, while following the discipline of software engineering best practice is decidedly not. We have developed a methodology, Live-Thru Case Histories, for overcoming this aversion, and have found it to be very productive in a pilot study. We are developing the methodology further for use both at universities and in industry.

Software projects often fail because too many software folks think that software engineering *process* is just bureaucracy. Often software people lack software engineering education, or when they had it, they fought it. To compound the problem, they do not accept software best practices. Our challenge is to overcome the natural biases of software professionals.

We tried lecturing on case histories of failed software projects, but these lectures and associated readings only convince many students of others' stupidity. They do not internalize the lessons. Others intellectually accept the existence of the problems, but just reading about them does not convert many at the gut level where one sits up, takes notice, and does things differently.

At the gut level, successful software project managers instinctively anticipate problems and take steps to avoid them. The challenge is to educate software people so that they do not have to hone their instincts through the "school of hard knocks."

Our approach is to force students to live through specific case histories, each one chosen to get across a small number of important issues. This method works. Students internalize the software engineering lessons and follow best practices in their next projects to avoid the traps they experienced.

Here is how the approach works. First, select a set of software process issues. These are the ones we chose for our first live-through case history:

- The need to have close customer/user relations.
- The need for up-to-date documentation throughout the life of the project.
- The need to identify risks and to develop contingency plans.
- The need to account for human foibles.

Second, choose a case history based on a project facing these challenges. Do not give students the entire case history up front; rather, give them the same problem as the actual developers who executed the case history faced. Give the students no more information about the problem than the original developers had at the start. You may simplify information to ease understanding.

Background

Computer science is the study of the technology (state-of-the-art) involved in the development of computer software. As it is usually taught, computer science deals with *programming in the small*, i.e., one-person or few-person software projects. Software engineering, on the other hand, is the study of the method or process (state-of-the-practice) whereby production software is developed – *programming in the large*. State-of-the-practice includes both engineering practices and project management or group dynamic processes.

Typical computer science programs offer a software engineering or senior project course as a capstone. Due to the very different natures of technology vs. method/process, and because computer science students are typically technology-oriented and process-averse, the typical software engineering course reaches far fewer future software developers than suits the best interests of either the students or the software industry. Thus we developed a novel instructional method, the Live-Thru Case History method for addressing this problem. We have developed a first live-through case history and have used it successfully in the first few weeks of a two-semester undergraduate software-engineering course.

The result was that students were shocked into an awareness of the issues and how to deal with them in only six weeks of twice-weekly class meetings. One class meeting each week was devoted to individual unstructured project meetings, and the other to lectures on software engineering topics, including other case histories.

Conducting the Case History

There would be just one live-through case history in our senior project course, so we had to choose one that would achieve the greatest effect in the limited time available. We chose the case history of a brief development project that one of the authors worked on in 1985 as a public service project. The project was automating an elementary school library's manual system for generating overdue-book notices.

The class of 40 students was divided randomly into four equal-size development teams. Students were given the same details possessed by the original software developers in the case history. The instructor role-played the customer, the school librarian, and was available to respond to students' questions, both in class and by e-mail. Students were told that the customer would evaluate their work exactly as it would be evaluated in the real world.

Results

As is frequently the case in real software development projects, the overdue book notice project had a hidden requirement. That requirement was so obvious to the customer that she failed to mention it; overdue notices must be sorted first by teacher name, then for each teacher by class, and finally, within each class by student's family name. The system analyst rejected the real software system when she first saw it. The original developers failed to elicit the hidden make-or-break requirement, and thus failed to satisfy it. Each of the student teams fell into this same trap thus learning the lesson of the need to find any hidden requirements.

Students also learned of the need for high-quality documentation and contingency planning due to the real-world phenomenon of attrition through illness, death, relocation, etc. At the project midpoint, students were rotated. A student from each team judged by the instructor to be the team's strongest developer and another chosen randomly were removed from the team and reassigned to a different team.

To evaluate each team's success in adapting to the simulated attrition, students were asked to describe what they would have done differently after the case study project was complete. About 75 percent of the students mentioned the importance of up-to-date documentation. Nearly 20 percent had developed insight into appropriate staff utilization, including the use of "understudies" and preparing for incorporating new team members. They demonstrated the learned value of these processes.

An evaluation of how well the students internalized the need for solid requirements engineering was performed at the end of the live-through case history. Students completed a written exam based on another case history that included a more difficult requirements engineering problem than that of the overdue book notice project. About 75 percent of the students demonstrated they had mastered the notion of hidden requirements, and about 33 percent showed they had achieved reasonable competence in requirements engineering; about 10 percent showed extremely keen insight into the problem.

The innovative process of live-through case histories is more effective than the traditionally taught software engineering course. In it, students were given lectures, homework, and exams based on a well-respected software engineering text. Then they were asked to develop a project. However when they approached the project, they could not readily apply the learned techniques. Once they understood the need for the processes, they relearned them as they tried to apply them.◆

Directions

The authors request that those teaching software engineering use the Live-Thru Case Histories in their courses and report on the results. These materials are available at www.njcse.org/Projects/Live_Thru_Case_Histories/Materials_For_Live_Thru_Case_Histories.htm, along with a complete paper describing the live through approach in detail.

Please participate in gathering data to support or refute the claims in this paper. It is our intent to use the experience of instructors in several venues to make anecdotal conclusions more meaningful and perhaps statistically significant. We invite those who agree with us to join a consortium for the purpose of creating additional case histories and helping to refine the process.

The Great Learning Process (Confucius)

Things being investigated (piloted), knowledge became complete.

Their knowledge being complete, their process was updated.

Their process being updated, their practices were cultivated.

Their practices being cultivated, their projects were regulated.

Their projects being regulated, their organizations were rightly governed.

Their organizations being rightly governed, the whole corporation was made tranquil and prosperous.

Adapted slightly by and with apologies from Tim Powell

About the Authors

David Klappholz has 27 years of experience teaching computer science and performing and supervising technology research sponsored by such organizations as National Science Foundation, Department of Energy, IBM Research, and The New Jersey Commission on Science and Technology. He has been on the computer science faculties of Columbia University and Polytechnic University, and is currently on the computer science faculty at Stevens Institute of Technology, in Hoboken, N.J., and associate director of the New Jersey Center for Software Engineering.

Department of Computer Science
Stevens Institute of Technology
Castle Point Station
Hoboken, NJ 07030
Phone: (908) 464-0805
E-mail: d.klappholz@worldnet.att.net

Lawrence Bernstein is a former vice president of AT&T where he managed small-, medium-, and large-scale software projects, both commercial and military, for 35 years. He is a Fellow of both the Institute of Electrical and Electronics Engineers and the Association for Computing Machinery. He is currently senior industry professor of Software Engineering at Stevens Institute of Technology, in Hoboken, N.J., and director of the New Jersey Center for Software Engineering.

Department of Computer Science
Stevens Institute of Technology
Castle Point Station
Hoboken, NJ 07030
Phone: (973)258-9213
E-mail: lbernstein@worldnet.att.net

"Before software can be reusable it first has to be usable."
— Ralph Johnson